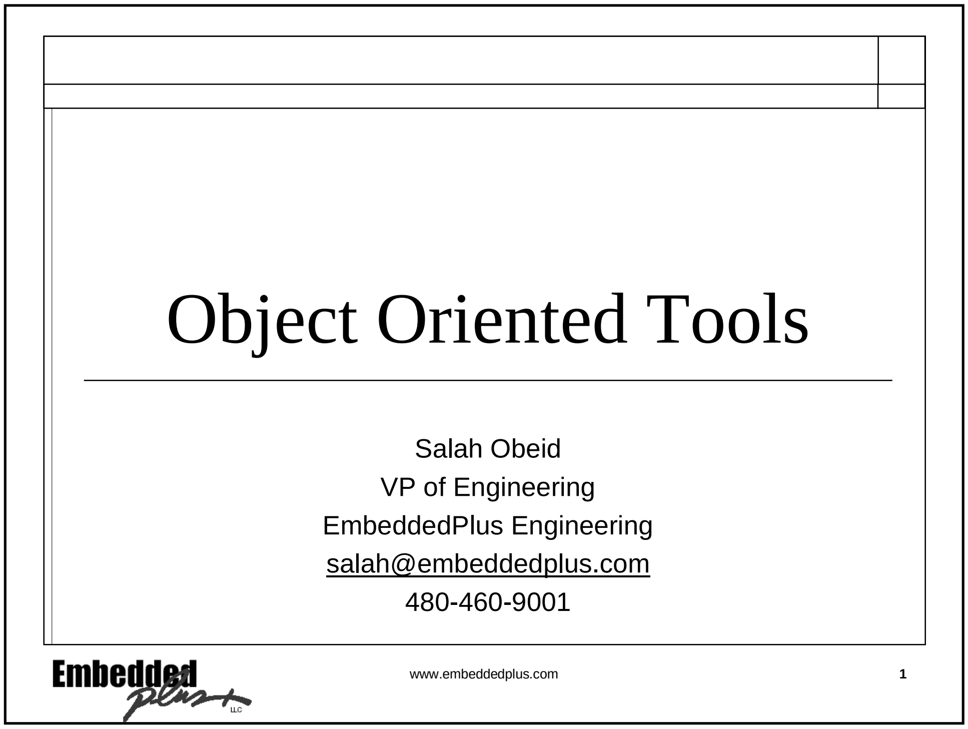
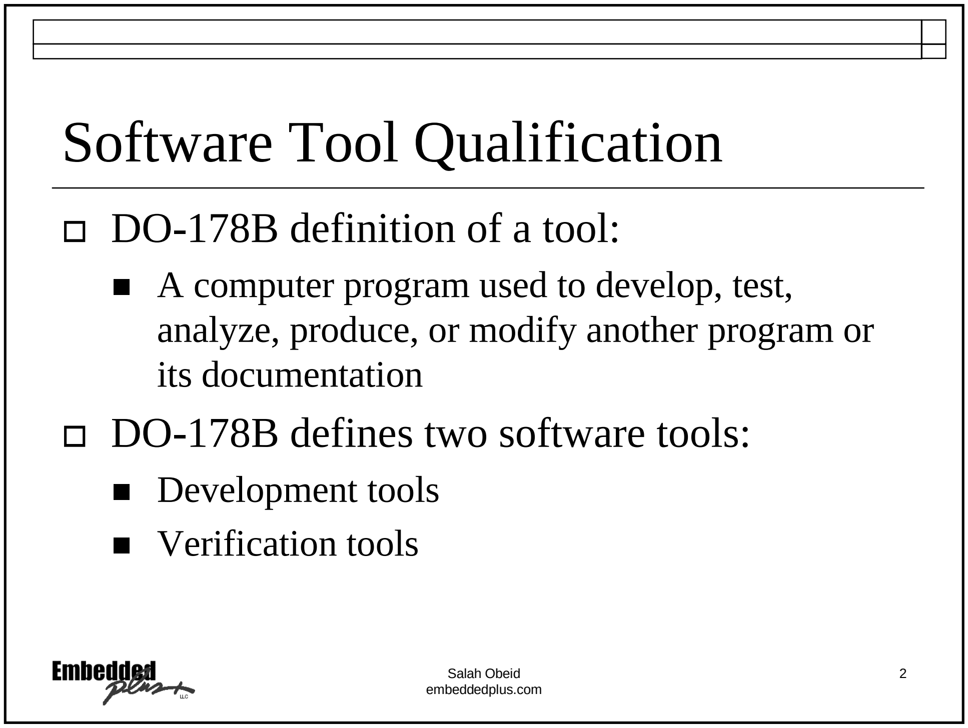


Object Oriented Tools

Salah Obeid
VP of Engineering
EmbeddedPlus Engineering
salah@embeddedplus.com
480-460-9001

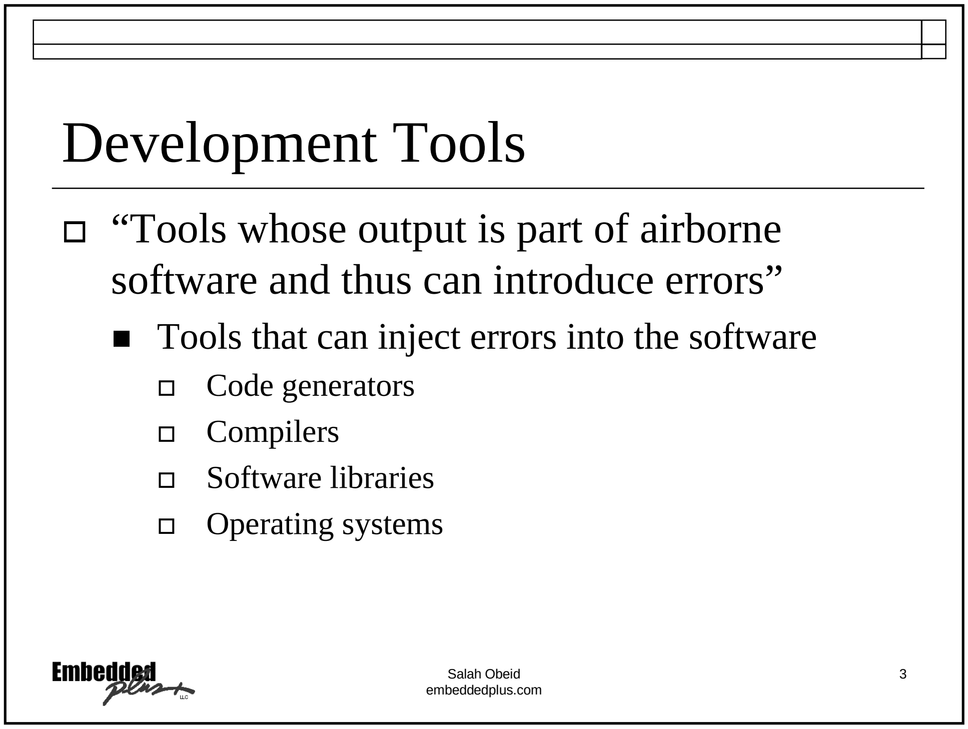
www.embeddedplus.com 1



Software Tool Qualification

- DO-178B definition of a tool:
  - A computer program used to develop, test, analyze, produce, or modify another program or its documentation
- DO-178B defines two software tools:
  - Development tools
  - Verification tools

Salah Obeid
embeddedplus.com 2



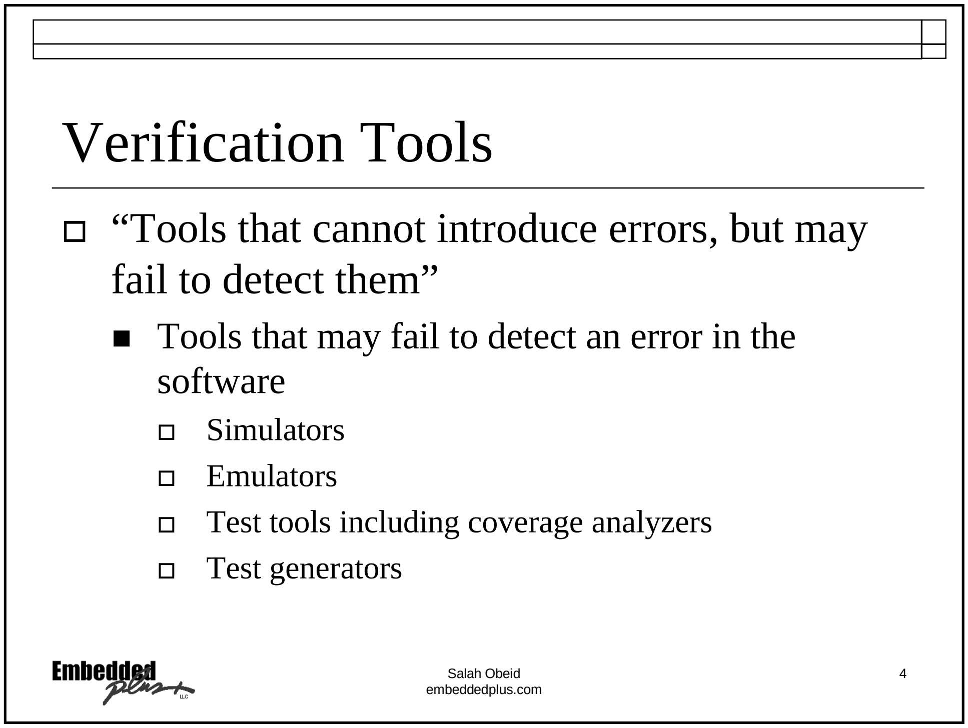
Development Tools

- □ "Tools whose output is part of airborne software and thus can introduce errors"
  - ■ Tools that can inject errors into the software
    - □ Code generators
    - □ Compilers
    - □ Software libraries
    - □ Operating systems

Embedded plus LLC

Salah Obeid
embeddedplus.com

3



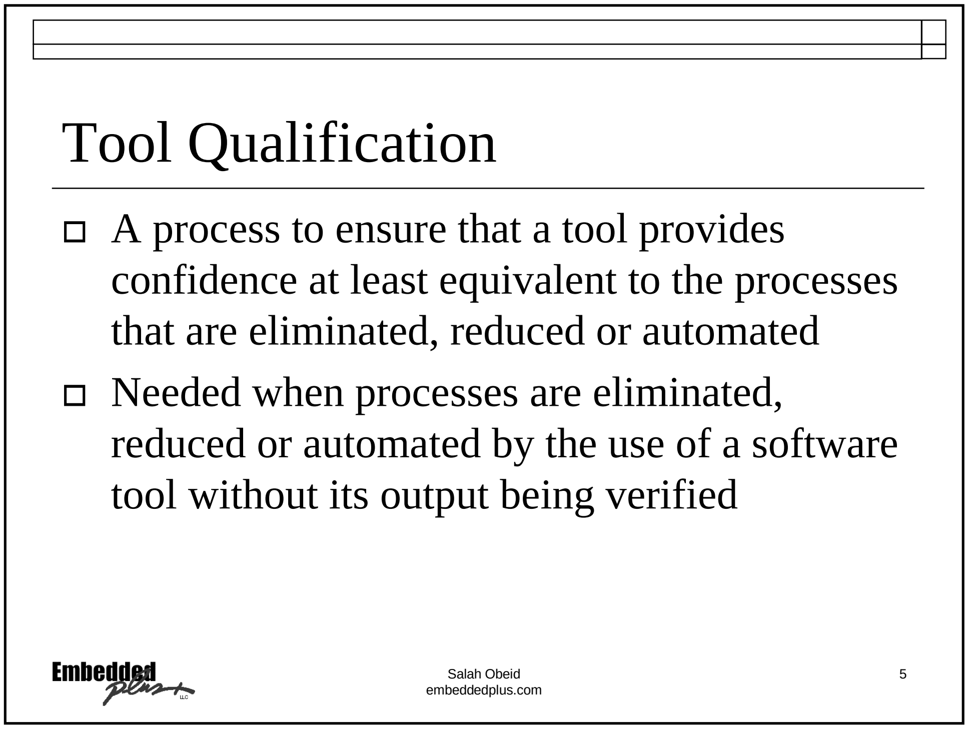
Verification Tools

- □ "Tools that cannot introduce errors, but may fail to detect them"
  - ■ Tools that may fail to detect an error in the software
    - □ Simulators
    - □ Emulators
    - □ Test tools including coverage analyzers
    - □ Test generators

Embedded plus LLC

Salah Obeid
embeddedplus.com

4

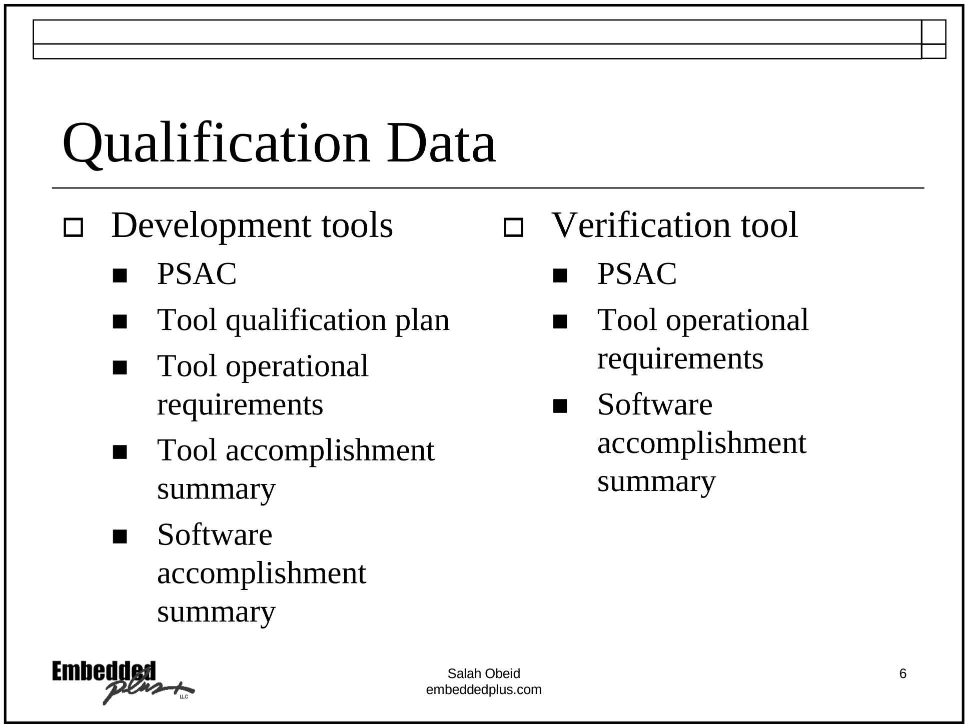


Tool Qualification

- □ A process to ensure that a tool provides confidence at least equivalent to the processes that are eliminated, reduced or automated
- □ Needed when processes are eliminated, reduced or automated by the use of a software tool without its output being verified

Salah Obeid
embeddedplus.com

5

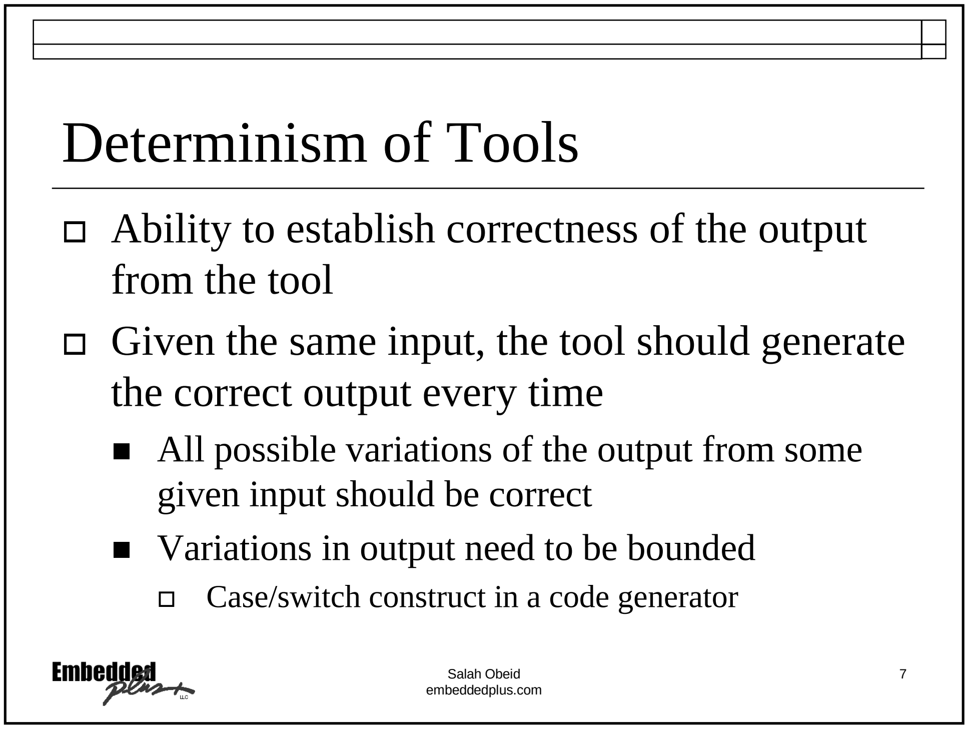


Qualification Data

- □ Development tools
  - ■ PSAC
  - ■ Tool qualification plan
  - ■ Tool operational requirements
  - ■ Tool accomplishment summary
  - ■ Software accomplishment summary
- □ Verification tool
  - ■ PSAC
  - ■ Tool operational requirements
  - ■ Software accomplishment summary

Salah Obeid
embeddedplus.com

6



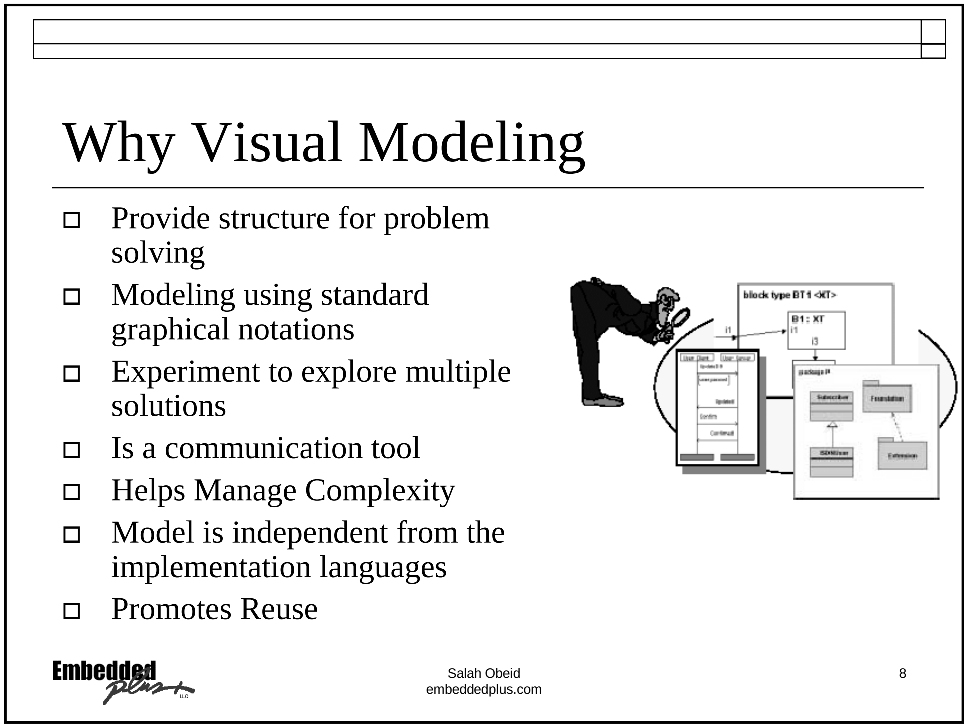
Determinism of Tools

- □ Ability to establish correctness of the output from the tool
- □ Given the same input, the tool should generate the correct output every time
  - ■ All possible variations of the output from some given input should be correct
  - ■ Variations in output need to be bounded
    - □ Case/switch construct in a code generator

Embedded plus LLC

Salah Obeid
embeddedplus.com

7



Why Visual Modeling

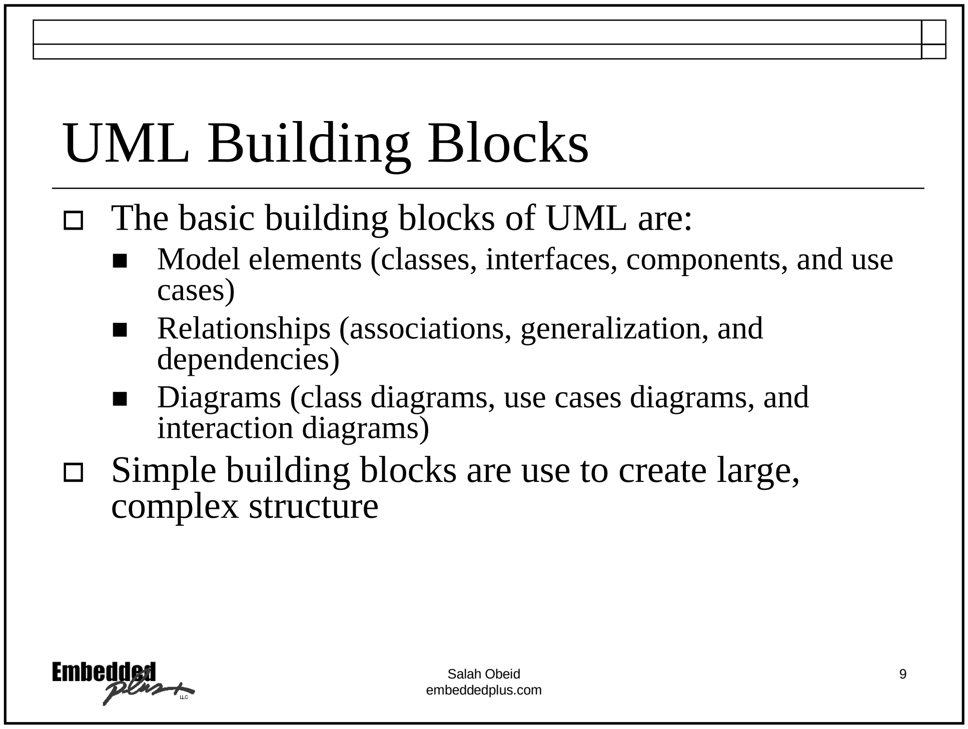
- □ Provide structure for problem solving
- □ Modeling using standard graphical notations
- □ Experiment to explore multiple solutions
- □ Is a communication tool
- □ Helps Manage Complexity
- □ Model is independent from the implementation languages
- □ Promotes Reuse

block type BT1 <XT>
B1 : XT
i1
i1
i3
package P
Subscriber
Foundation
ISDNUser
Extension
Update
Confirm
Continue

Embedded plus LLC

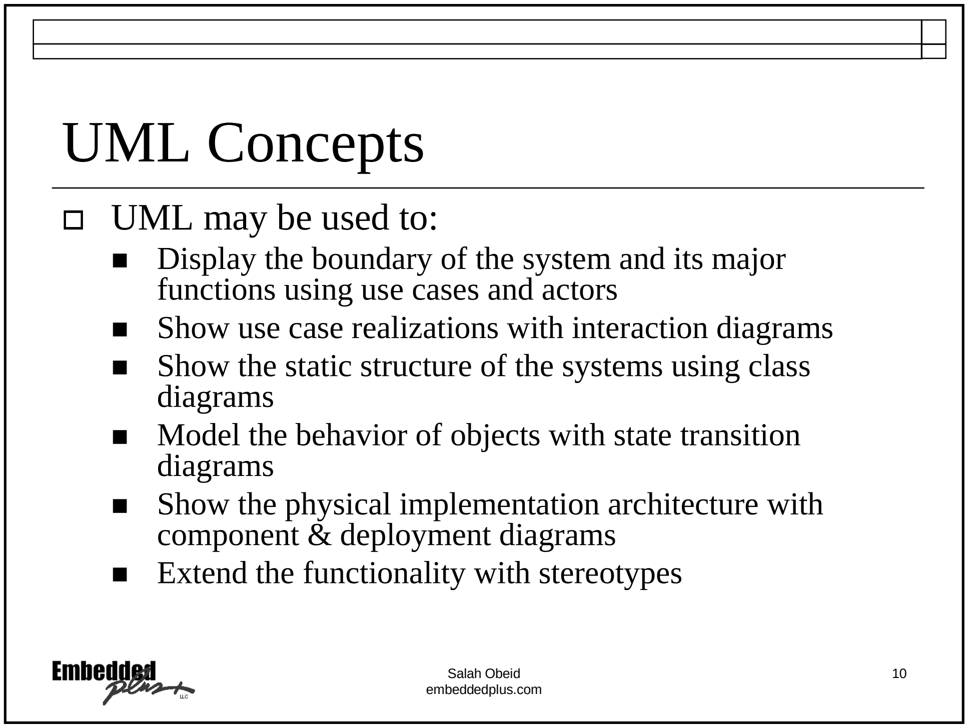
Salah Obeid
embeddedplus.com

8



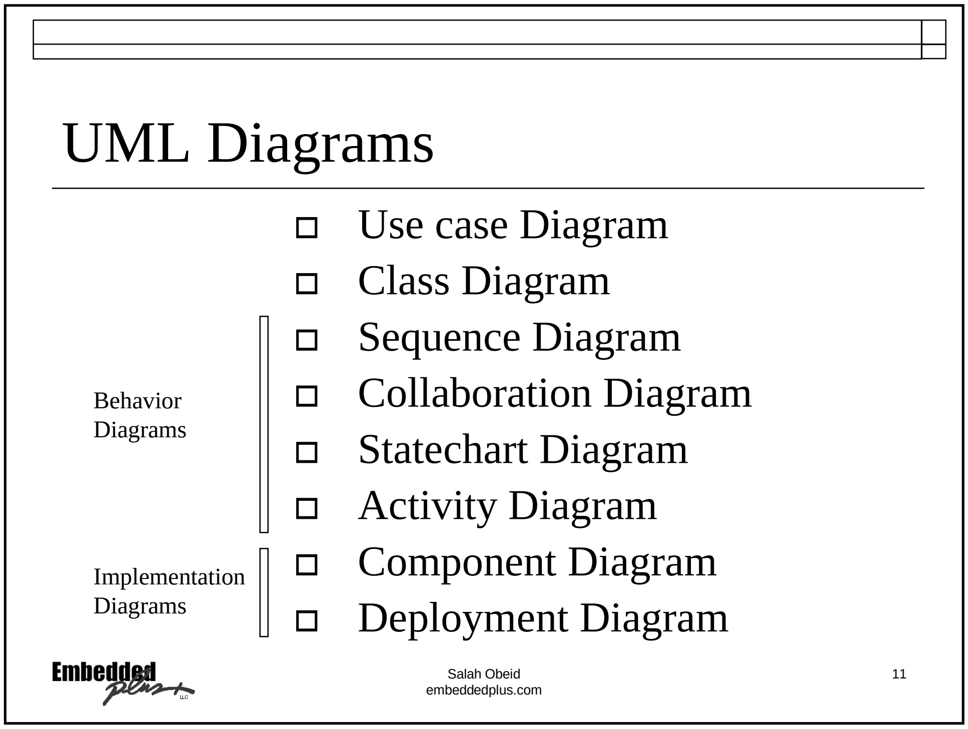
UML Building Blocks

- □ The basic building blocks of UML are:
  - ■ Model elements (classes, interfaces, components, and use cases)
  - ■ Relationships (associations, generalization, and dependencies)
  - ■ Diagrams (class diagrams, use cases diagrams, and interaction diagrams)
- □ Simple building blocks are use to create large, complex structure



UML Concepts

- □ UML may be used to:
  - ■ Display the boundary of the system and its major functions using use cases and actors
  - ■ Show use case realizations with interaction diagrams
  - ■ Show the static structure of the systems using class diagrams
  - ■ Model the behavior of objects with state transition diagrams
  - ■ Show the physical implementation architecture with component & deployment diagrams
  - ■ Extend the functionality with stereotypes



UML Diagrams

- □ Use case Diagram
- □ Class Diagram

Behavior Diagrams

- □ Sequence Diagram
- □ Collaboration Diagram
- □ Statechart Diagram
- □ Activity Diagram

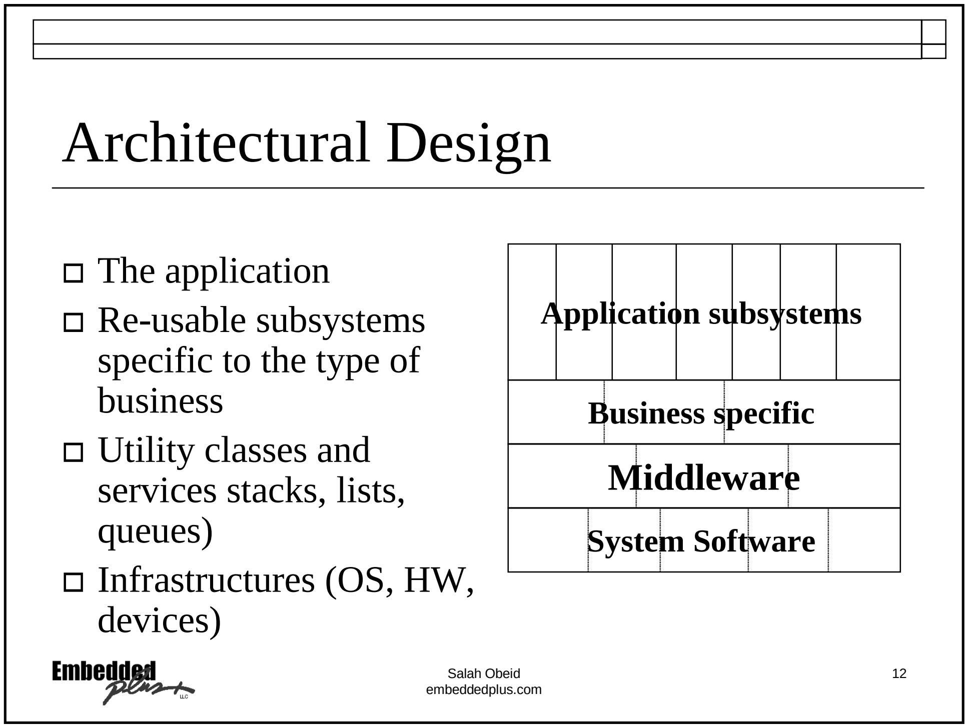
Implementation Diagrams

- □ Component Diagram
- □ Deployment Diagram

Embedded plus LLC

Salah Obeid
embeddedplus.com

11



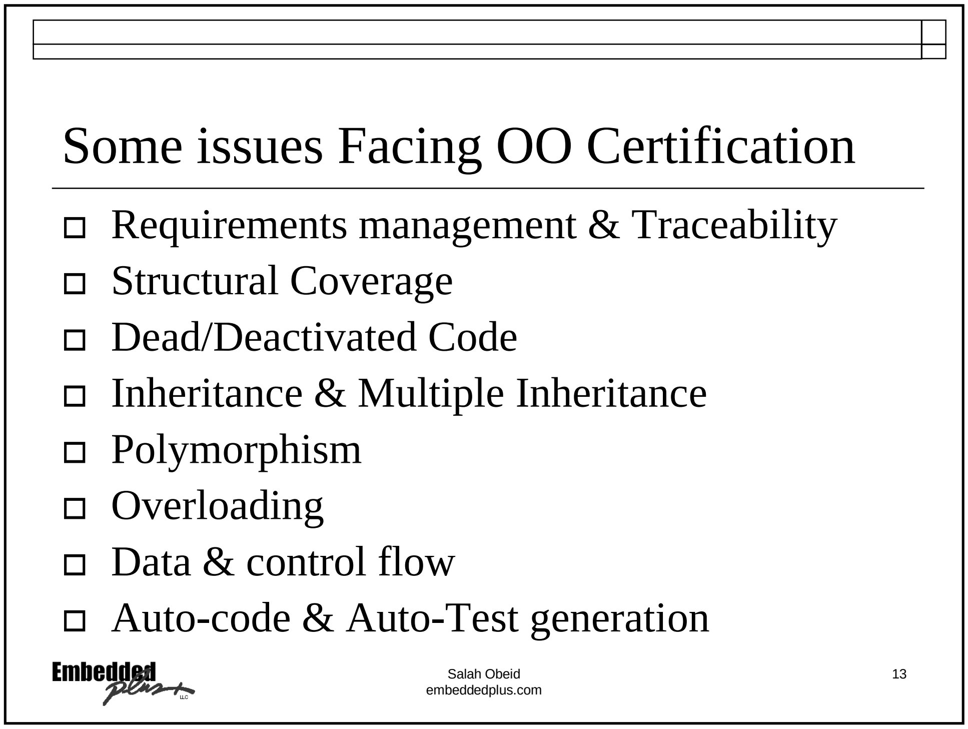
Architectural Design

- □ The application
- □ Re-usable subsystems specific to the type of business
- □ Utility classes and services stacks, lists, queues)
- □ Infrastructures (OS, HW, devices)

Embedded plus LLC

Salah Obeid
embeddedplus.com

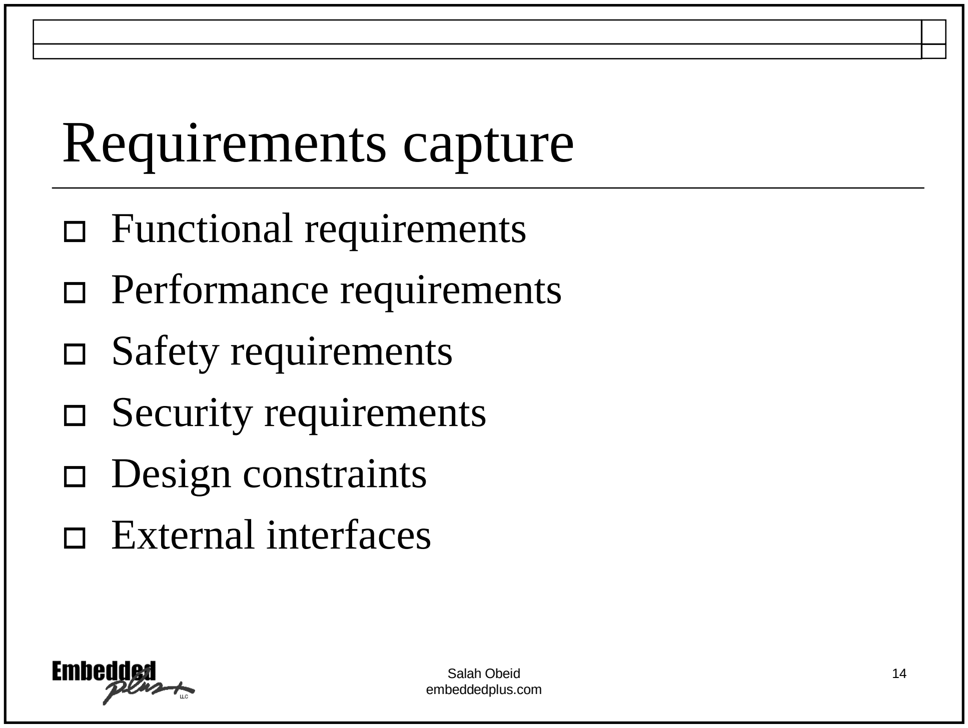
12



Some issues Facing OO Certification

- □ Requirements management & Traceability
- □ Structural Coverage
- □ Dead/Deactivated Code
- □ Inheritance & Multiple Inheritance
- □ Polymorphism
- □ Overloading
- □ Data & control flow
- □ Auto-code & Auto-Test generation

Salah Obeid
embeddedplus.com
13

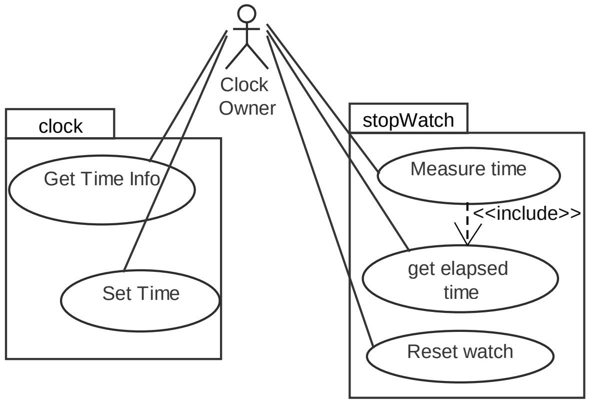


Requirements capture

- □ Functional requirements
- □ Performance requirements
- □ Safety requirements
- □ Security requirements
- □ Design constraints
- □ External interfaces

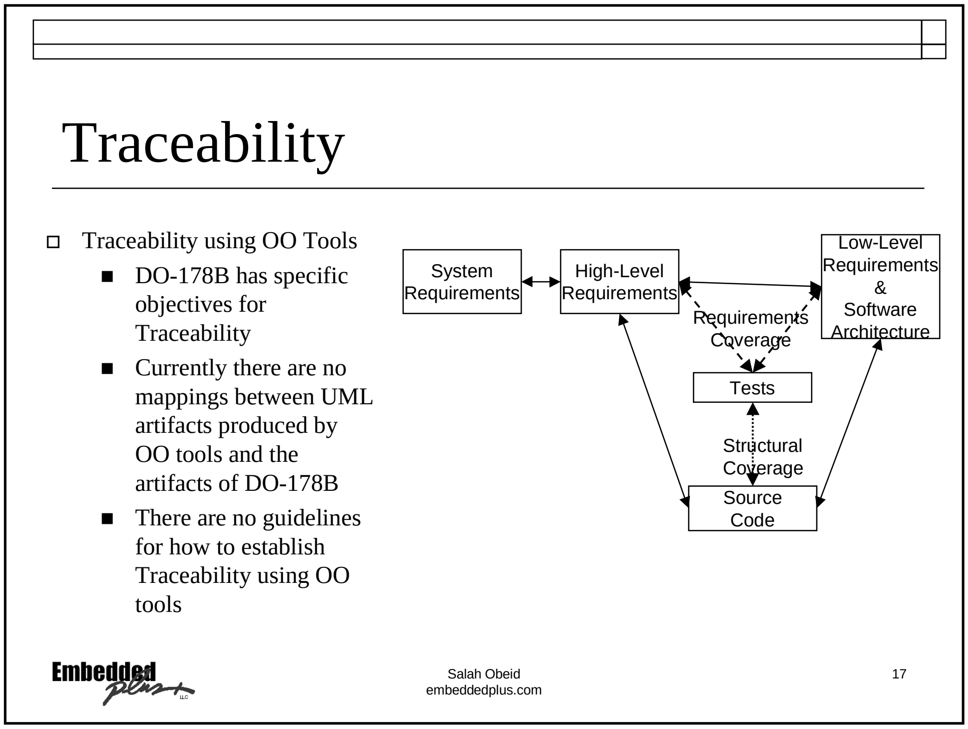
Salah Obeid
embeddedplus.com
14

Use Cases



Use Case Description

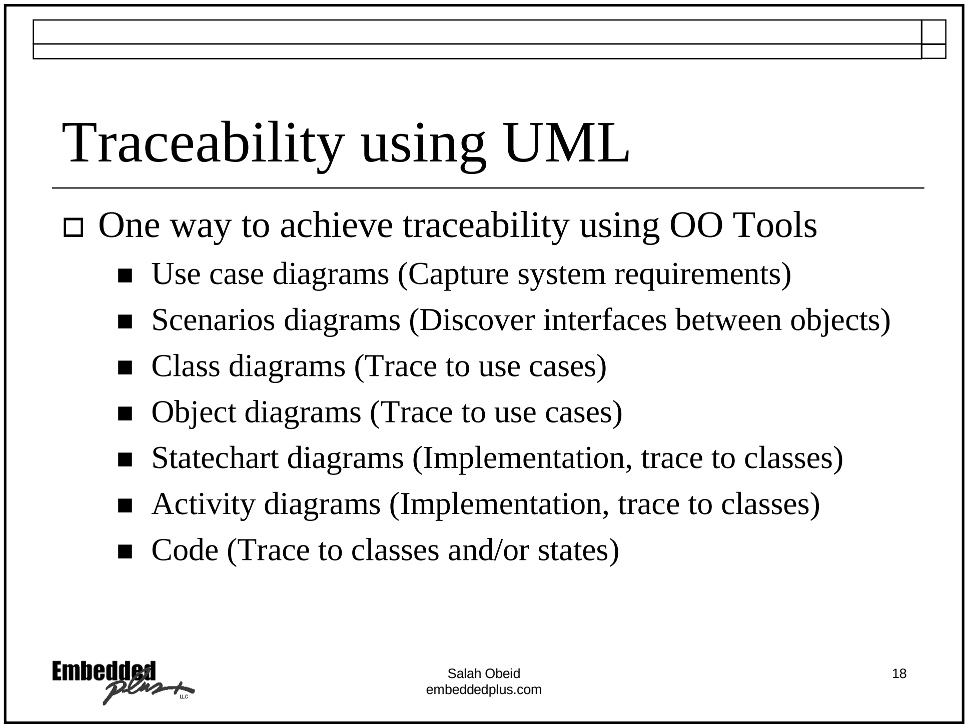
- □ Name:
- □ Actors:
- □ Pre-conditions:
- □ Begins:
- □ Description:
- □ Ends:
- □ Exceptions:
- □ Post condition:



Traceability

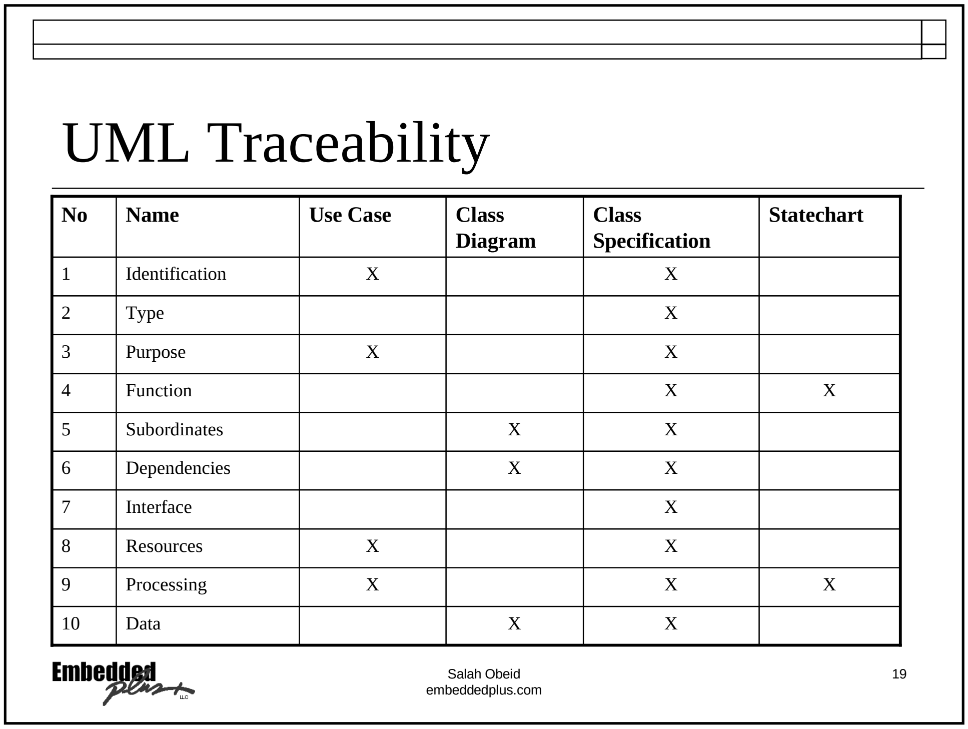
- Traceability using OO Tools
  - DO-178B has specific objectives for Traceability
  - Currently there are no mappings between UML artifacts produced by OO tools and the artifacts of DO-178B
  - There are no guidelines for how to establish Traceability using OO tools

System Requirements
High-Level Requirements
Low-Level Requirements & Software Architecture
Requirements Coverage
Tests
Structural Coverage
Source Code



Traceability using UML

- One way to achieve traceability using OO Tools
  - Use case diagrams (Capture system requirements)
  - Scenarios diagrams (Discover interfaces between objects)
  - Class diagrams (Trace to use cases)
  - Object diagrams (Trace to use cases)
  - Statechart diagrams (Implementation, trace to classes)
  - Activity diagrams (Implementation, trace to classes)
  - Code (Trace to classes and/or states)

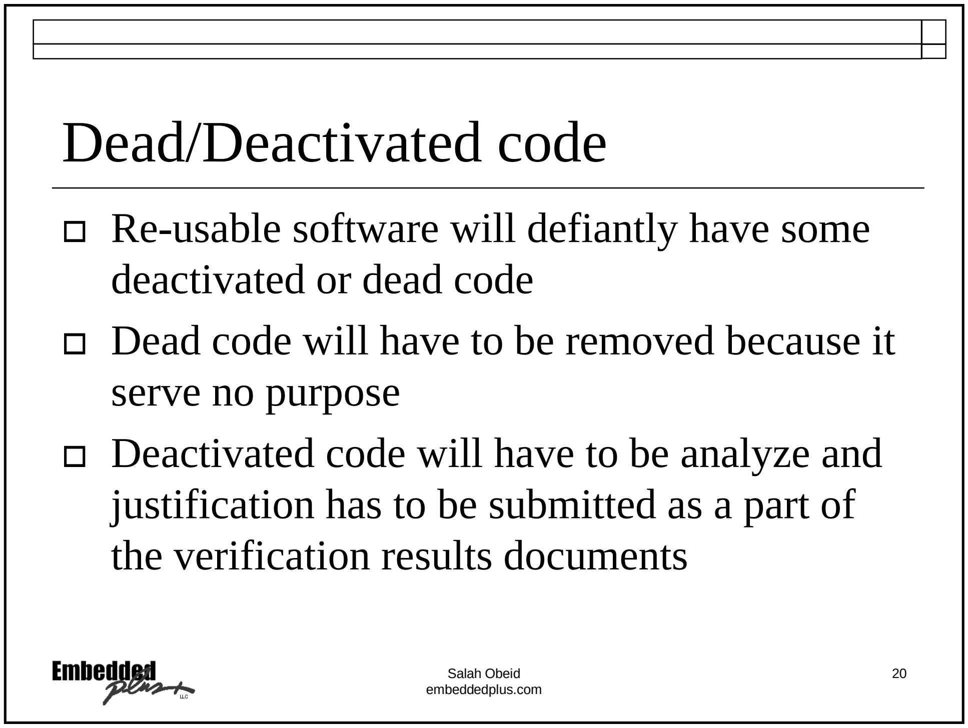


UML Traceability

No	Name	Use Case	Class Diagram	Class Specification	Statechart
1	Identification	X		X	
2	Type			X	
3	Purpose	X		X	
4	Function			X	X
5	Subordinates		X	X	
6	Dependencies		X	X	
7	Interface			X	
8	Resources	X		X	
9	Processing	X		X	X
10	Data		X	X	

Salah Obeid
embeddedplus.com

19

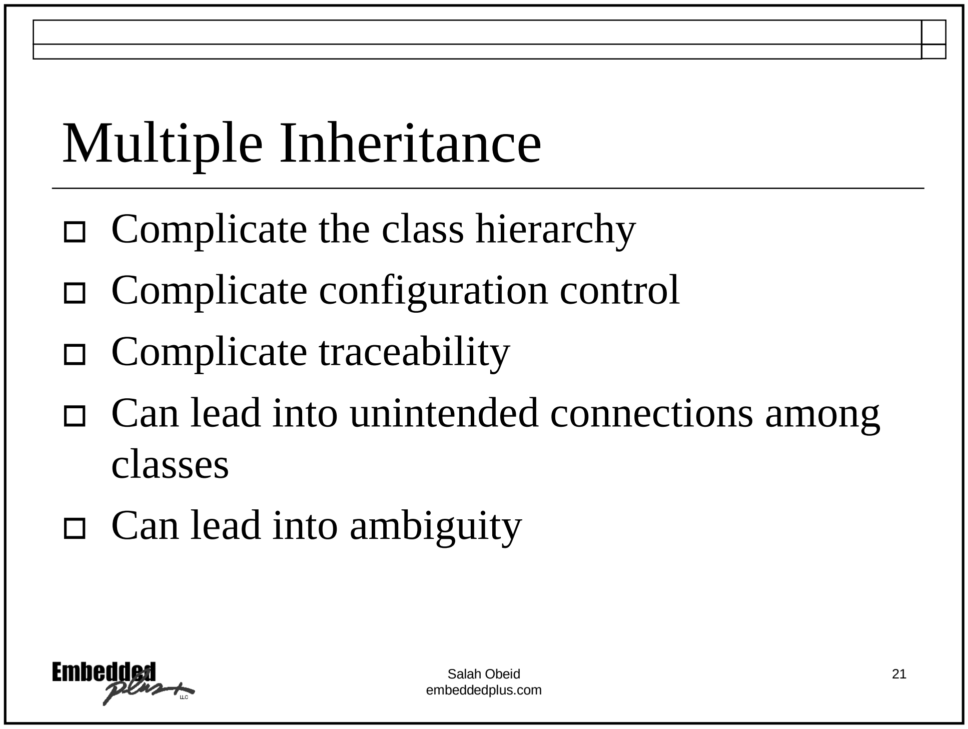


Dead/Deactivated code

- Re-usable software will defiantly have some deactivated or dead code
- Dead code will have to be removed because it serve no purpose
- Deactivated code will have to be analyze and justification has to be submitted as a part of the verification results documents

Salah Obeid
embeddedplus.com

20

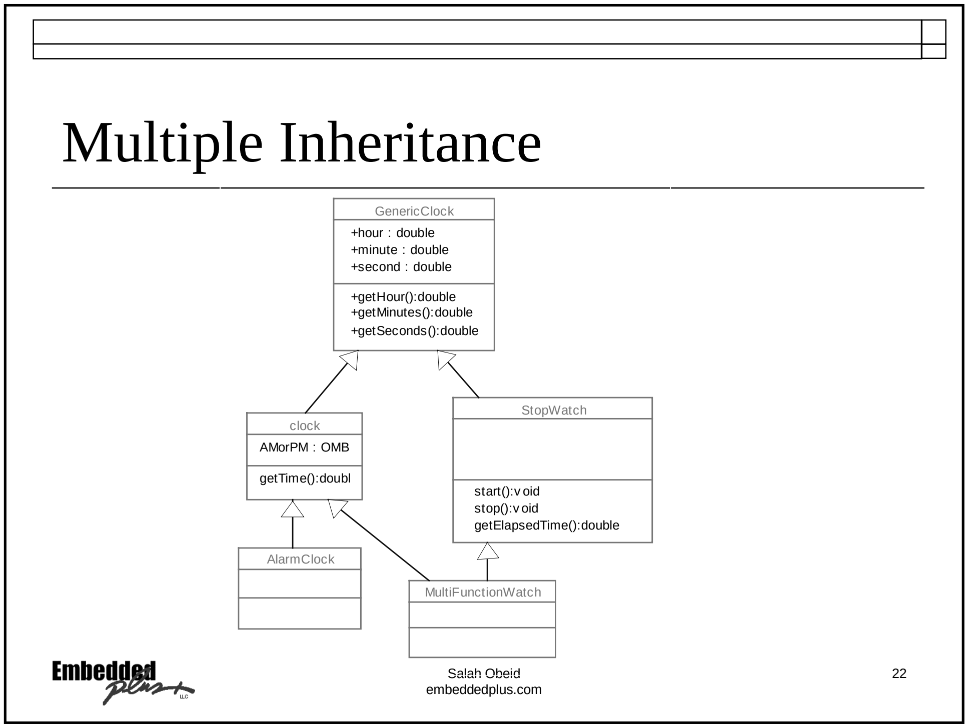


Multiple Inheritance

- □ Complicate the class hierarchy
- □ Complicate configuration control
- □ Complicate traceability
- □ Can lead into unintended connections among classes
- □ Can lead into ambiguity

Salah Obeid
embeddedplus.com

21



Multiple Inheritance

GenericClock
+hour : double
+minute : double
+second : double
+getHour():double
+getMinutes():double
+getSeconds():double

clock
AMorPM : OMB
getTime():doubl

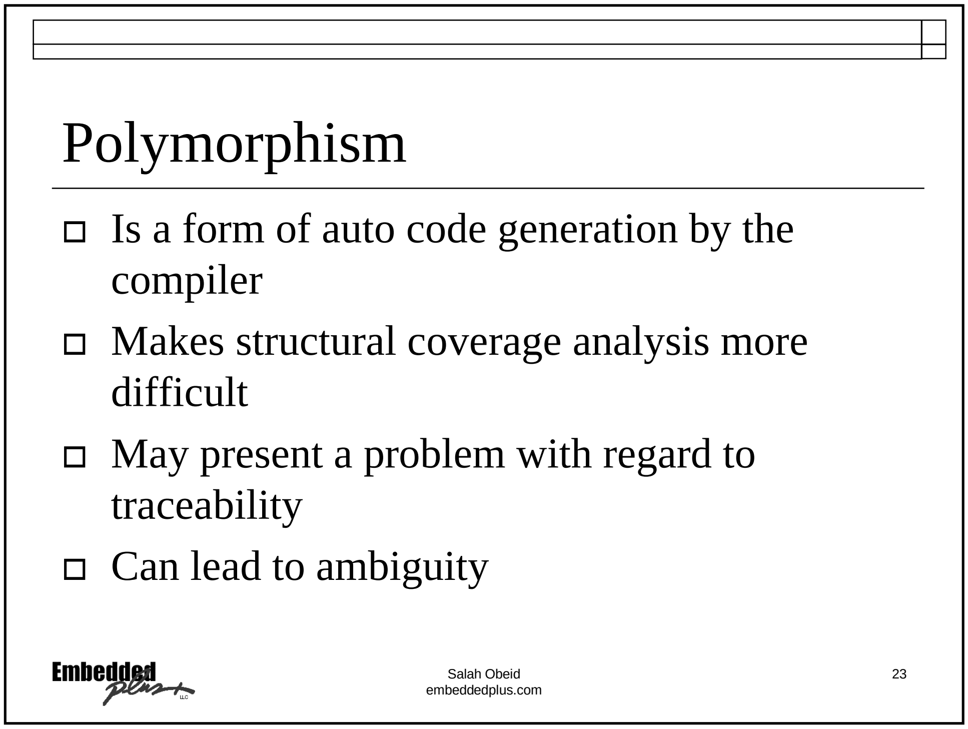
StopWatch
start():void
stop():void
getElapsedTime():double

AlarmClock

MultiFunctionWatch

Salah Obeid
embeddedplus.com

22

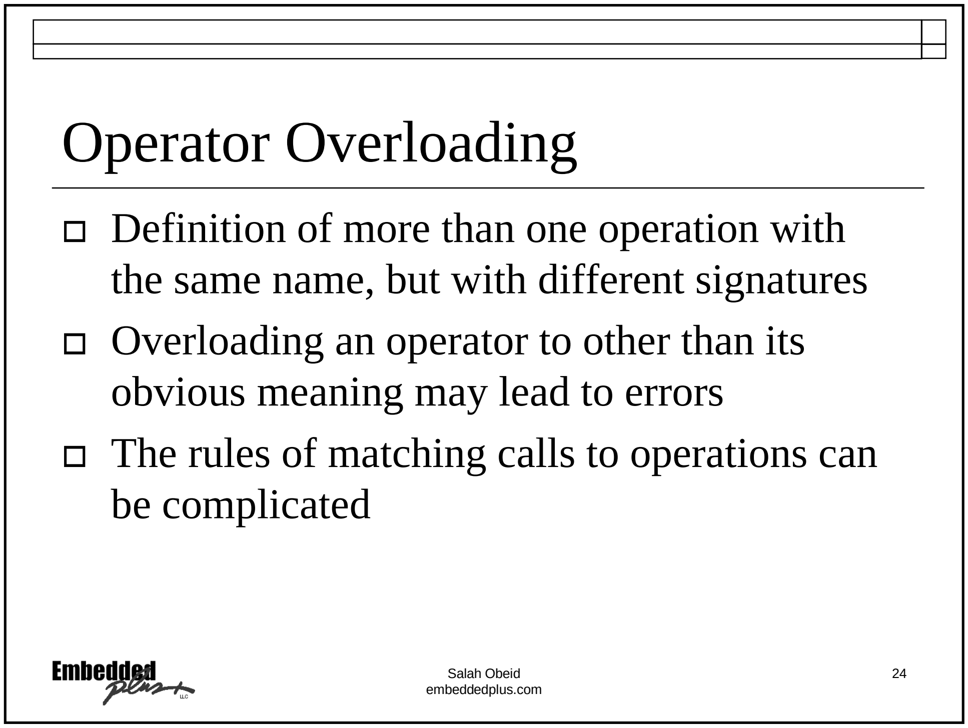


Polymorphism

- □ Is a form of auto code generation by the compiler
- □ Makes structural coverage analysis more difficult
- □ May present a problem with regard to traceability
- □ Can lead to ambiguity

Salah Obeid
embeddedplus.com

23

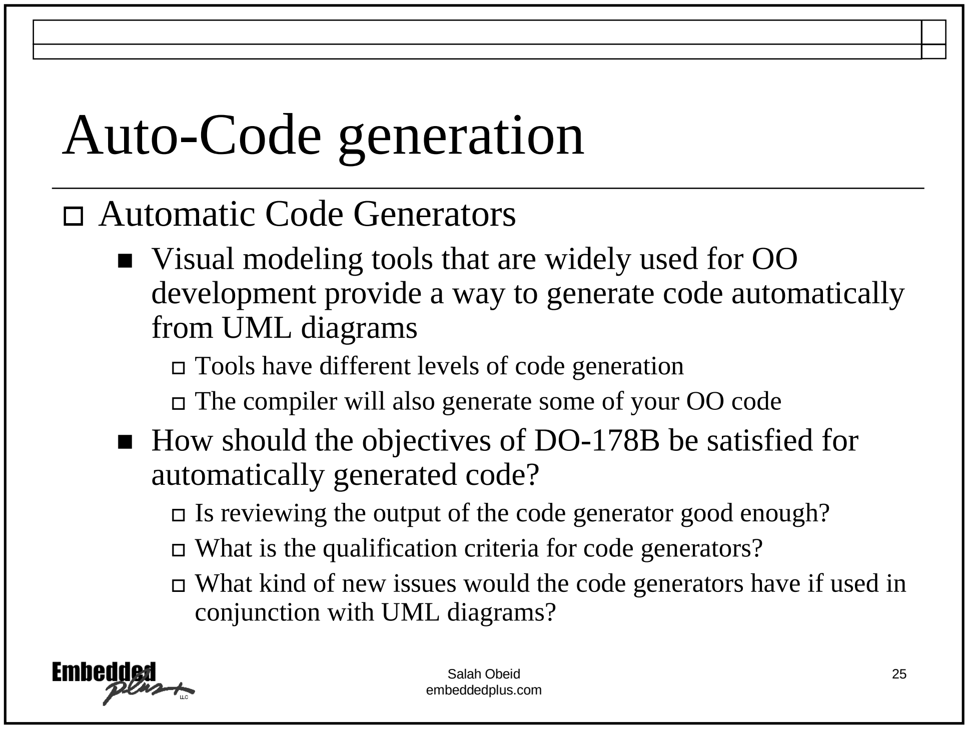


Operator Overloading

- □ Definition of more than one operation with the same name, but with different signatures
- □ Overloading an operator to other than its obvious meaning may lead to errors
- □ The rules of matching calls to operations can be complicated

Salah Obeid
embeddedplus.com

24

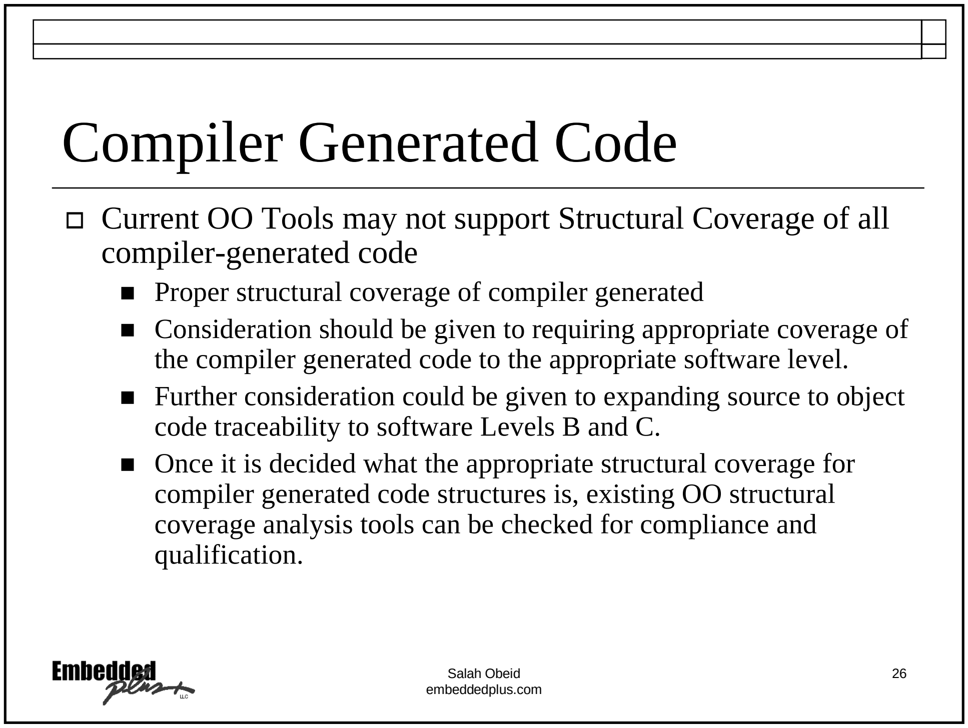


Auto-Code generation

- □ Automatic Code Generators
  - ■ Visual modeling tools that are widely used for OO development provide a way to generate code automatically from UML diagrams
    - □ Tools have different levels of code generation
    - □ The compiler will also generate some of your OO code
  - ■ How should the objectives of DO-178B be satisfied for automatically generated code?
    - □ Is reviewing the output of the code generator good enough?
    - □ What is the qualification criteria for code generators?
    - □ What kind of new issues would the code generators have if used in conjunction with UML diagrams?

Salah Obeid
embeddedplus.com

25

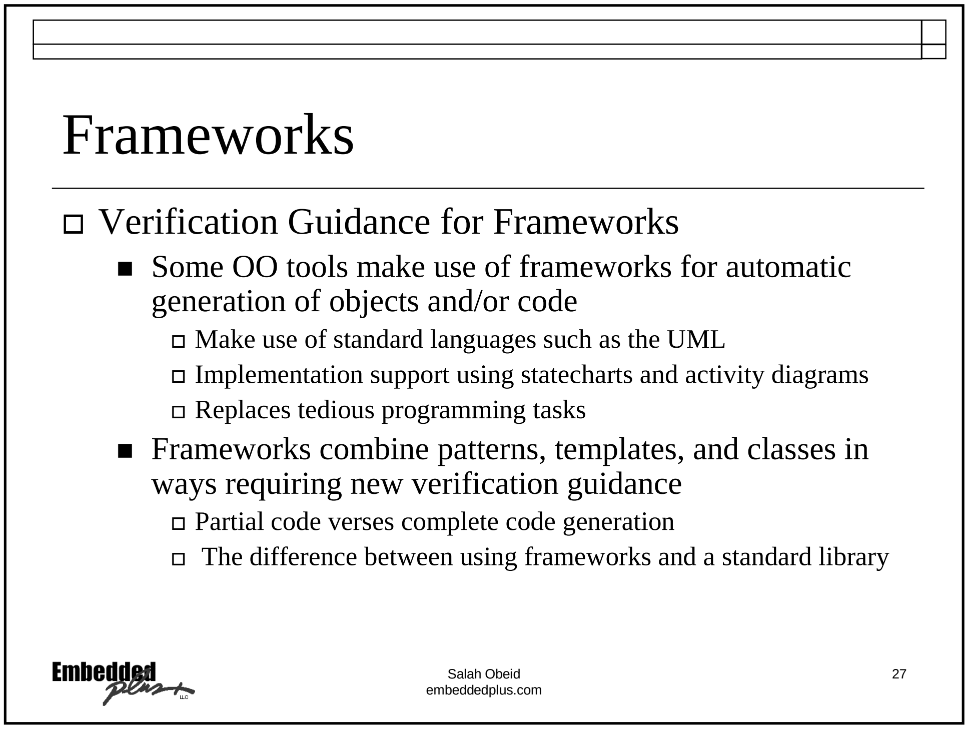


Compiler Generated Code

- □ Current OO Tools may not support Structural Coverage of all compiler-generated code
  - ■ Proper structural coverage of compiler generated
  - ■ Consideration should be given to requiring appropriate coverage of the compiler generated code to the appropriate software level.
  - ■ Further consideration could be given to expanding source to object code traceability to software Levels B and C.
  - ■ Once it is decided what the appropriate structural coverage for compiler generated code structures is, existing OO structural coverage analysis tools can be checked for compliance and qualification.

Salah Obeid
embeddedplus.com

26

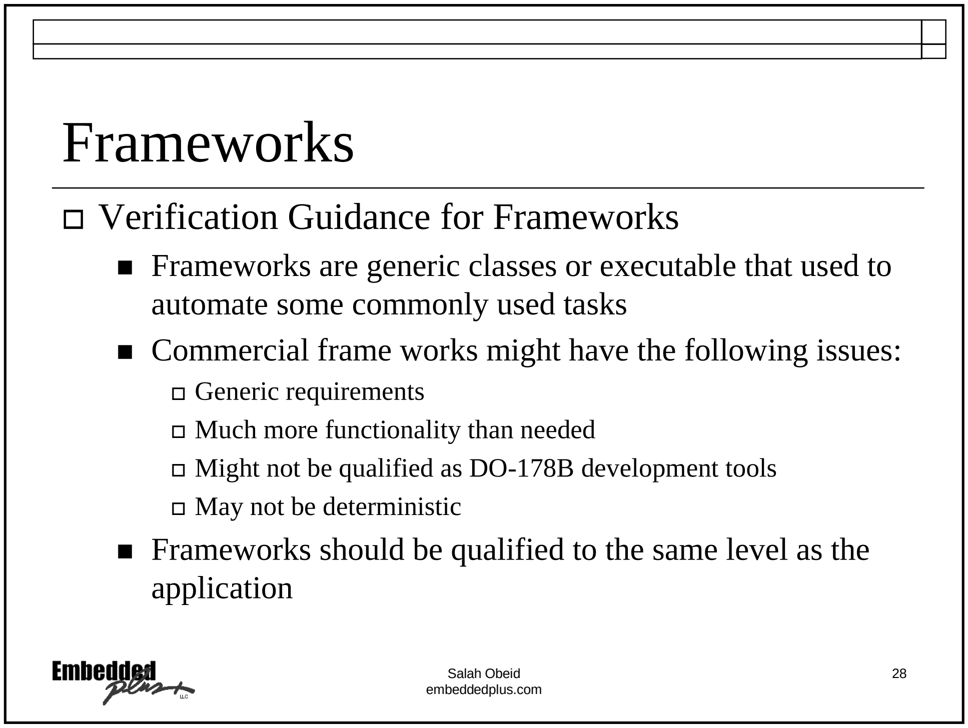


Frameworks

- □ Verification Guidance for Frameworks
  - ■ Some OO tools make use of frameworks for automatic generation of objects and/or code
    - □ Make use of standard languages such as the UML
    - □ Implementation support using statecharts and activity diagrams
    - □ Replaces tedious programming tasks
  - ■ Frameworks combine patterns, templates, and classes in ways requiring new verification guidance
    - □ Partial code verses complete code generation
    - □ The difference between using frameworks and a standard library

Salah Obeid
embeddedplus.com

27

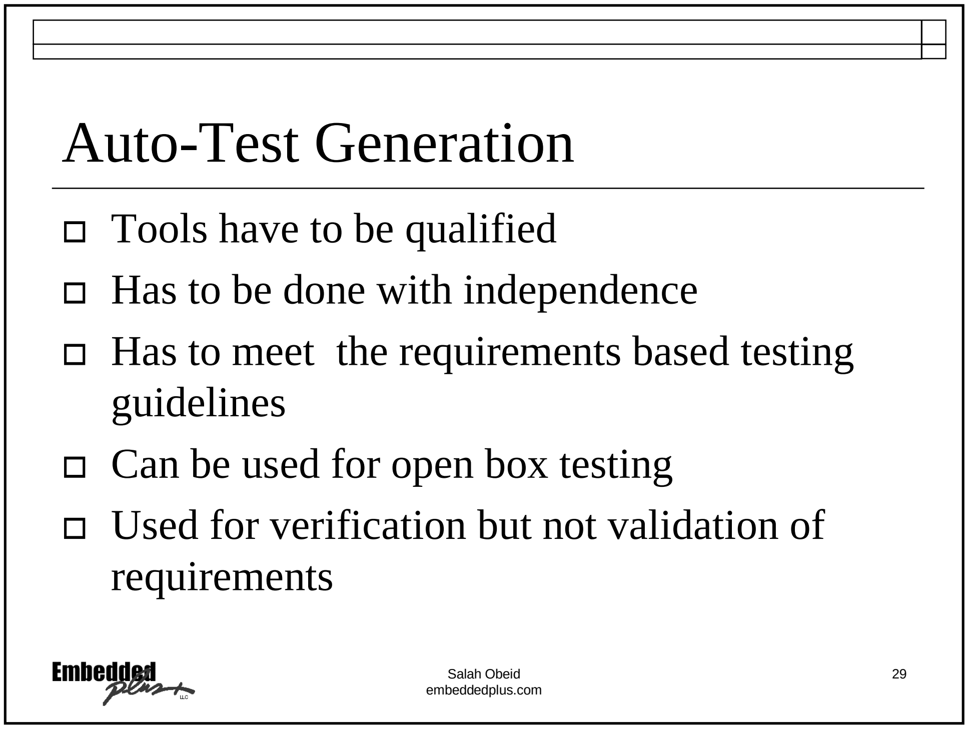


Frameworks

- □ Verification Guidance for Frameworks
  - ■ Frameworks are generic classes or executable that used to automate some commonly used tasks
  - ■ Commercial frame works might have the following issues:
    - □ Generic requirements
    - □ Much more functionality than needed
    - □ Might not be qualified as DO-178B development tools
    - □ May not be deterministic
  - ■ Frameworks should be qualified to the same level as the application

Salah Obeid
embeddedplus.com

28

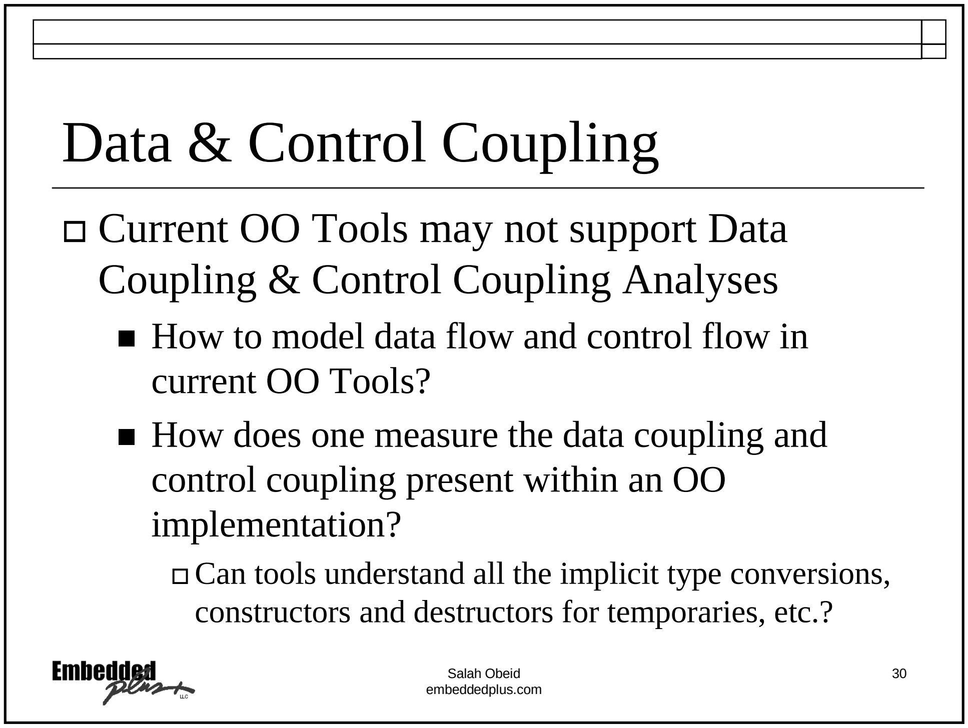


Auto-Test Generation

- □ Tools have to be qualified
- □ Has to be done with independence
- □ Has to meet the requirements based testing guidelines
- □ Can be used for open box testing
- □ Used for verification but not validation of requirements

Salah Obeid
embeddedplus.com

29

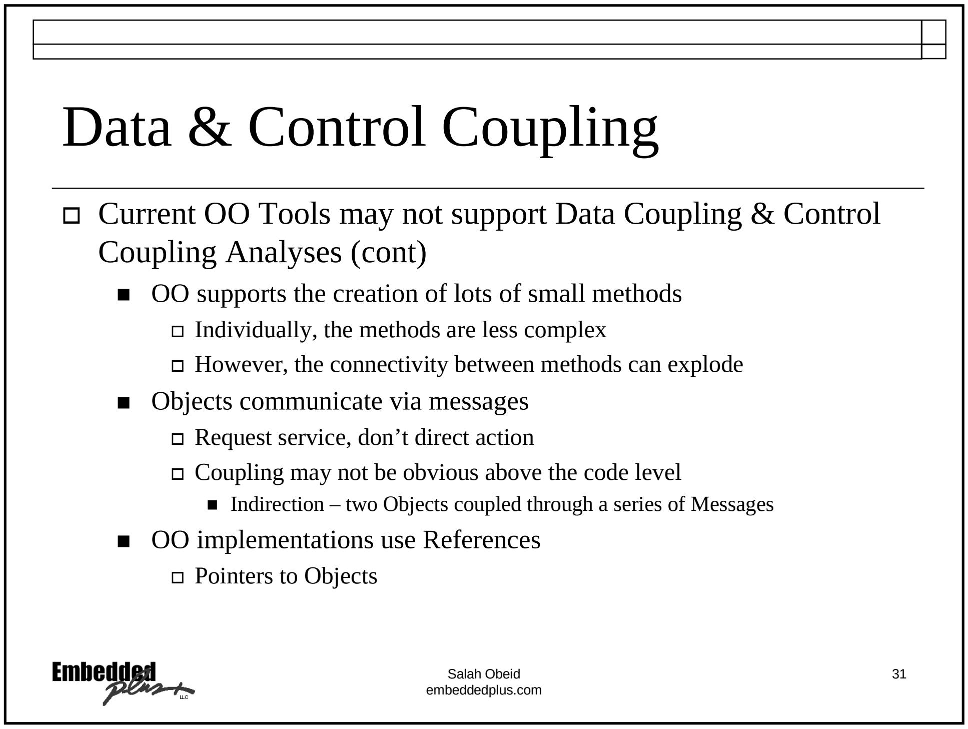


Data & Control Coupling

- □ Current OO Tools may not support Data Coupling & Control Coupling Analyses
  - ■ How to model data flow and control flow in current OO Tools?
  - ■ How does one measure the data coupling and control coupling present within an OO implementation?
    - □ Can tools understand all the implicit type conversions, constructors and destructors for temporaries, etc.?

Salah Obeid
embeddedplus.com

30

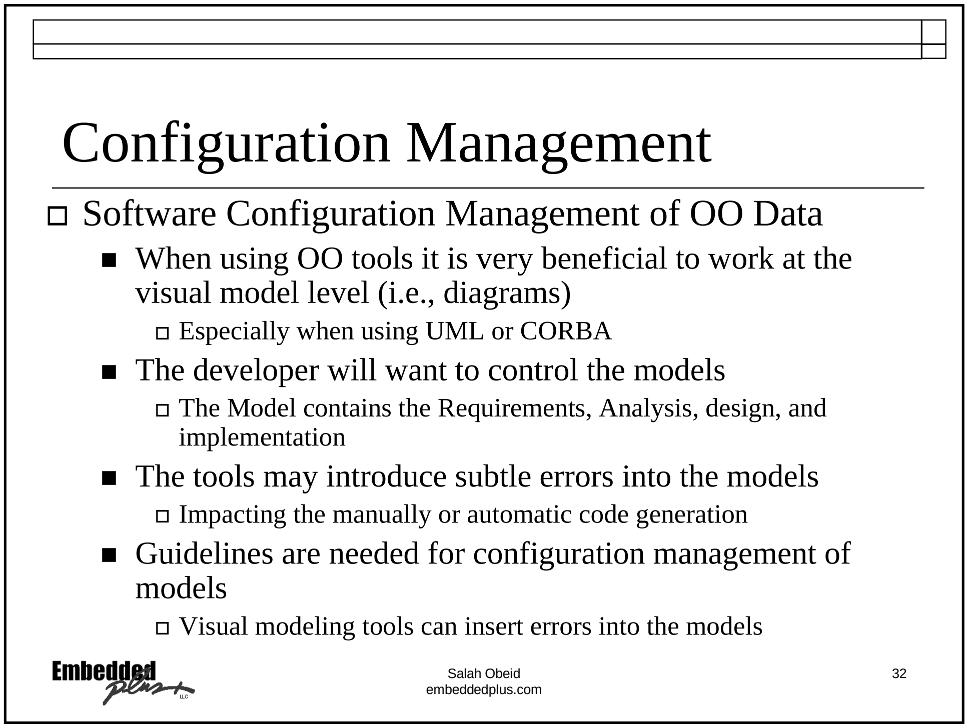


Data & Control Coupling

- □ Current OO Tools may not support Data Coupling & Control Coupling Analyses (cont)
  - ■ OO supports the creation of lots of small methods
    - □ Individually, the methods are less complex
    - □ However, the connectivity between methods can explode
  - ■ Objects communicate via messages
    - □ Request service, don’t direct action
    - □ Coupling may not be obvious above the code level
      - ■ Indirection – two Objects coupled through a series of Messages
  - ■ OO implementations use References
    - □ Pointers to Objects

Salah Obeid
embeddedplus.com

31

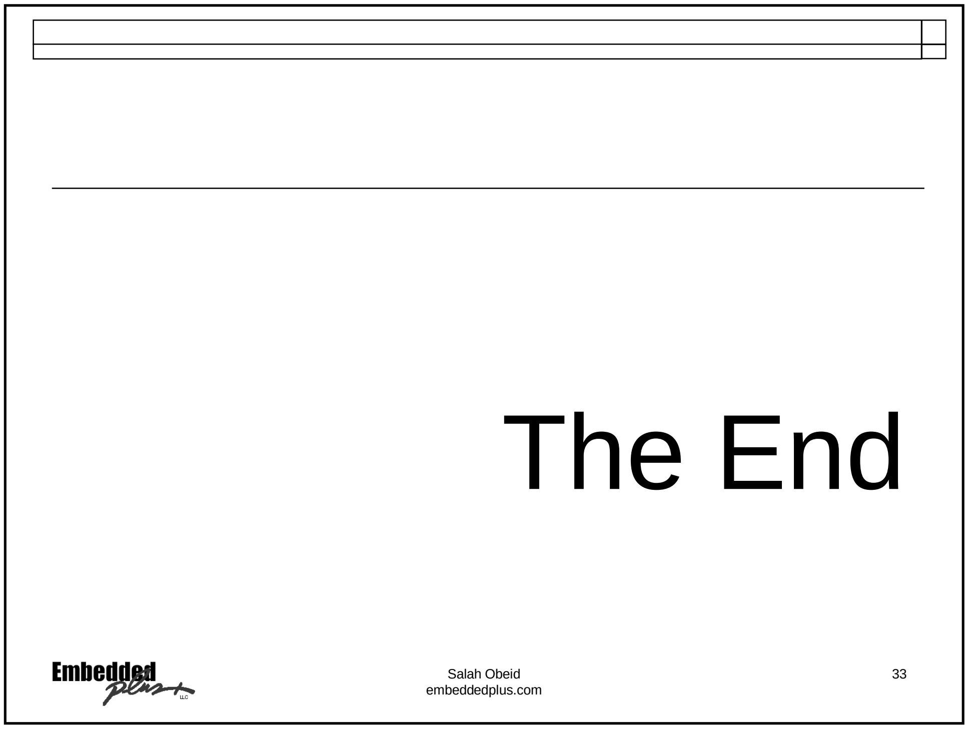


Configuration Management

- □ Software Configuration Management of OO Data
  - ■ When using OO tools it is very beneficial to work at the visual model level (i.e., diagrams)
    - □ Especially when using UML or CORBA
  - ■ The developer will want to control the models
    - □ The Model contains the Requirements, Analysis, design, and implementation
  - ■ The tools may introduce subtle errors into the models
    - □ Impacting the manually or automatic code generation
  - ■ Guidelines are needed for configuration management of models
    - □ Visual modeling tools can insert errors into the models

Salah Obeid
embeddedplus.com

32



The End

Salah Obeid
embeddedplus.com

33